**과목명: 시스템프로그래밍**

**담당 교수 명: 김 지 환**

**2분반**

**<<Project #1>>**

**서강대학교 컴퓨터학과**

**20181618**

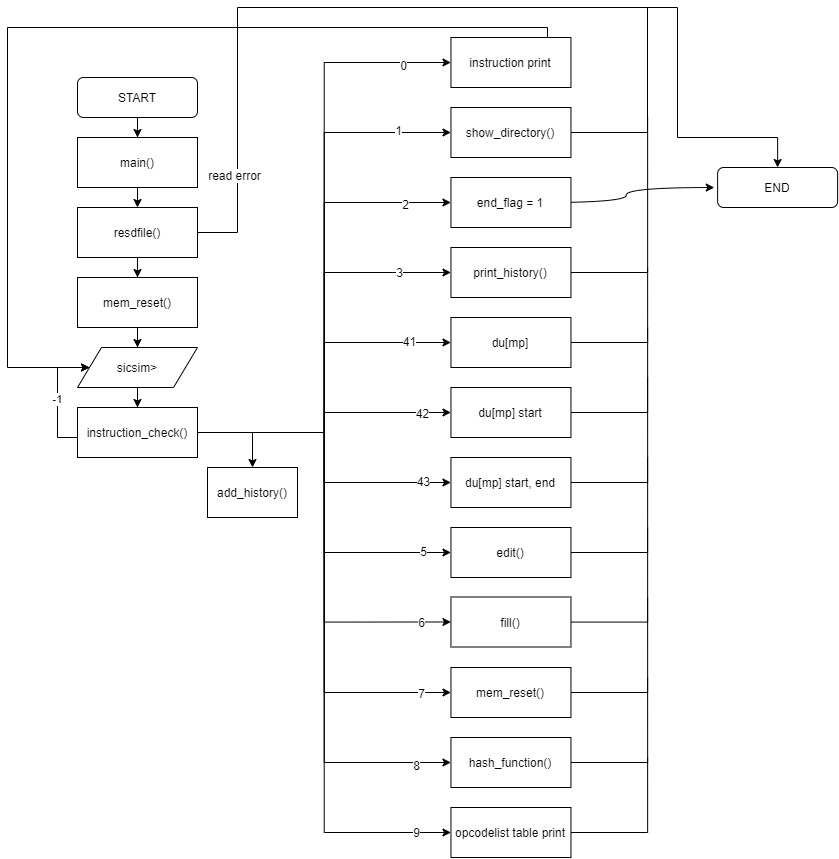
**김하늘**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. 모듈 이름 : main()
      1. 기능
      2. 사용 변수
   2. 모듈 이름 : readfile()
      1. 기능
      2. 사용 변수
   3. 모듈 이름 : hash\_function(char\* str)
      1. 기능
      2. 사용 변수
   4. 모듈 이름 : show\_directory()
      1. 기능
      2. 사용 변수
   5. 모듈 이름 : hexTodec(char\* str, char c, int key)
      1. 기능
      2. 사용 변수
   6. 모듈 이름 : mem\_reset()
      1. 기능
      2. 사용 변수
   7. 모듈 이름 : decTohec(int d)
      1. 기능
      2. 사용 변수
   8. 모듈 이름 : Dump(int key)
      1. 기능
      2. 사용 변수
   9. 모듈 이름 : edit()
      1. 기능
      2. 사용 변수
   10. 모듈 이름 : fill()
       1. 기능
       2. 사용 변수
   11. 모듈 이름 : boundary\_check(char\* str, int key)
       1. 기능
       2. 사용 변수
   12. 모듈 이름 : add\_history(char \*str)
       1. 기능
       2. 사용 변수
   13. 모듈 이름 : print\_history()
       1. 기능
       2. 사용 변수
   14. 모듈 이름 : free\_history()
       1. 기능
       2. 사용변수
   15. 모듈 이름 : free\_opcode()
       1. 기능
       2. 사용변수
   16. 모듈 이름 : instruction\_check(char\* inst)
       1. 기능
       2. 사용 변수
4. **전역 변수 정의**
   1. char m[65536][16]
   2. char start[10], end[10], address[10], value[10]
   3. char mne[10]
   4. int cur\_add
   5. int end\_flag
   6. int cur\_i
   7. hash\* h\_table[20]
   8. his\* history
5. **코드**
   1. 20181618.c
   2. 20181618.h
6. 프로그램 개요

이 프로그램은 이후 프로젝트와 연결되며, SIC/XE머신을 구현하기 위한 전 단계이다. 어셈블러, 링크, 로더들을 실행하게 될 셸(shell)과 컴파일을 통해서 만들어진 object코드가 실행되기 위해 필요한 메모리 공간, mnemonic과 대응되는 opcode 값을 테이블로 만들어 관련 명령어들을 구현하는 프로그램이다.

1. 프로그램 설명
   1. 프로그램 흐름도



프로그램의 Flow chart는 위와 같다. 프로그램이 실행되면 main함수가 시작되고, opcode.txt 파일을 읽는다. 이때 실패하면 프로그램이 종료되고, 성공할 경우 메모리를 리셋한 후 명령어를 입력받는다. 입력받은 명령어가 비정상적인 명령어이면 다시 입력을 받고, 정상적인 명령어면 history에 해당 명령어와 대응되는 값을 return한 뒤 수행한다. q 또는 quit이 입력될 경우 프로그램이 종료된다.

1. 모듈 정의
   1. 모듈 이름 : **main()**
      1. 기능

프로그램이 실행되면 우선 opcode.txt파일을 읽어 hash table에 저장한 후, 메모리를 리셋한다. 이후 종료 명령이 들어올 때 까지 명령어를 입력받으며 수행한다. 올바르지 않은 명령이 들어올 경우에도 에러 메세지를 출력한 후 다시 입력을 받는다. 또한 파일이 정상적으로 열리지 않으면 프로그램이 종료된다.

* + 1. 사용 변수

int i - 반복문의 인덱싱에 사용되는 변수이다.

int k\_inst - instruction\_check()의 반환값이 저장된다. switch문의 key값으로 사용된다.

hash\* temp - hash table의 리스트를 탐색하기 위한 temp 노드이다.

char inst[50] - 입력받은 명령어를 저장하는 정적배열이다.

char\* t\_str - hash fumction()에서 반환된 opcode가 저장된다.

* 1. 모듈 이름 : **readfile()**
     1. 기능

opcode.txt를 열어 한 줄씩 읽는다. 파일이 열리지 않을 경우 프로그램이 종료되며, 한 줄을 읽을 때마다 노드를 생성하여 명령어와 opcode를 저장하고 (입력받은 명령어의 첫번째 알파벳 - 'A')%21의 값을 인덱스로 하여 hash table에 맵핑한다.

* + 1. 사용 변수

FILE\* f - 파일을 열기위한 파일 포인터이다.

char buf[25] - 현재 읽고있는 파일의 한 줄이 저장되는 문자열이다.

int i, idx - 배열의 인덱싱에 사용된다.

int cnt - while문을 제어하기 위한 카운트 변수이다.

int key - hash func에 인자로 넘어가는 key값이다.

hash\* temp - hash table의 리스트를 탐색하기 위한 temp 노드이다.

* 1. 모듈 이름 : **hash\_function(char\* str)**
     1. 기능

입력받은 명령어를 인자로 받고, 관련한 key값을 계산하여 인덱스로 사용해 리스트를 탐색하여 일치하는 opcode를 찾아 반환한다. key값은 (입력받은 명령어의 첫번째 알파벳 - 'A')%21로 계산하며 매칭되는 opcode가 존재하지 않을 경우 “\0”를 반환한다.

* + 1. 사용 변수

char\* str -- 함수의 인자로, 명령어가 저장되어있다.

int key - 계산된 key값이 저장된다.

hash\* temp - hash table의 리스트를 탐색하기 위한 temp 노드이다.

* 1. 모듈 이름 : **show\_directory()**
     1. 기능

현재 디렉토리의 경로를 추출하여 그 경로를 통해 디렉토리를 열고, 폴더 안의 모든 파일을 탐색 및 출력한다. 디렉토리 파일은 '/'를 끝에 붙이고, 실행파일은 '\*'를 붙인다. 한 줄에 4개씩 출력한다.

* + 1. 사용 변수

DIR\* dir\_p - opendir()를 통해 얻은 디렉토리의 정보가 저장된다.

struct dirent\* entry - 파일 목록을 하나씩 가져올 때 사용되는 변수이다.

struct stat buf - 디렉토리 내의 파일을 구분하기 위한 정보가 저장된 구조체이다.

char\* path - 현재 디렉토리의 추출된 경로가 저장된다.

int cnt - 출력할때 사용되는 카운트 변수이다.

* 1. 모듈 이름 : **hexTodec(char\* str, char c, int key)**
     1. 기능

16진수의 문자열 또는 문자를 10진수로 바꾸어 반환한다.

* + 1. 사용 변수

char\* str -- 함수의 인자로, 16진수 문자열이 저장되어있다.

char c -- 함수의 인자로, 16진수 문자가 저장되어있다.

int key -- 함수의 인자로, 어떤 타입의 16진수를 10진수로 바꿀지 결정한다.

* 1. 모듈 이름 : **mem\_reset()**
     1. 기능

1byte크기의 가상 메모리를 0으로 초기화한다.

* + 1. 사용 변수

int i, j - 인덱싱에 사용된다.

* 1. 모듈 이름 : **decTohec(int d)**
     1. 기능

F 이하의 10진수 값을 16진수 char 문자로 바꾸어 반환한다.

* + 1. 사용 변수

int d -- 함수의 인자로, F 이하의 10진수 값이다.

* 1. 모듈 이름 : **Dump(int key)**
     1. 기능

dump 기능이 구현되어 있으며, 사용자가 지정한 범위의 메모리를 출력하는 함수이다. 인자로 들어오는 key값에 따라 [dump], [dump start], [dump start, end] 세 유형으로 나누어 처리한다.

* + 1. 사용 변수

int key -- 함수의 인자로, 어떤 유형의 명령어를 처리할 지에 대한 정보가 저장되어있다.

int i, j, a, temp, t, cnt- 반복문의 인덱스, temp 등에 사용되는 임시 변수이다.

int st\_i, st\_j, st\_n- 출력이 시작되는 메모리가 저장 및 갱신된다.

int show\_addr[5] - 출력될 주소가 저장 및 갱신된다.

char t\_str[4] - 주소값을 인덱싱하기위해 임시로 주소값이 16진수로 저장되는 배열이다.

int max - start, end 주소로부터 출력될 총 byte 수를 계산하여 저장한다.

* 1. 모듈 이름 : **edit()**
     1. 기능

edit address, value 명령어를 구현한 함수이다. 지정된 주소의 값을 변경한다.

* + 1. 사용 변수

int i, j, a, temp - 반복문의 인덱스, temp 등에 사용되는 임시 변수이다.

char t\_str[4] - 주소값을 인덱싱하기위해 임시로 주소값이 16진수로 저장되는 배열이다.

3. 10. 모듈 이름 : **fill()**

3. 10. 1. 기능

fill start, end, value를 구현한 함수이다. 지정된 주소의 범위만큼 저장된 값을 변경한다.

3. 10. 2. 사용 변수

int a, i, j, cnt, temp - 반복문의 인덱스, temp 등에 사용되는 임시 변수이다.

int st\_i, st\_j, st\_n- 출력이 시작되는 메모리가 저장 및 갱신된다.

char t\_str[4] - 주소값을 인덱싱하기위해 임시로 주소값이 16진수로 저장되는 배열이다.

int max - start, end 주소로부터 출력될 총 byte 수를 계산하여 저장한다.

3. 11. 모듈 이름 : **boundary\_check(char\* str, int key)**

3. 11. 1. 기능

입력받는 주소값 및 데이터의 범위가 정상적인지 체크하는 함수이다. 입력받는 인자는 모두 16진수이므로, 0~F인지 검사한 뒤 FFFFF 또는 FF를 넘어가지 않는지를 고려한다. 정상적인 값일 경우 0을 반환하고 그렇지 않을경우 1을 반환한다.

3. 11. 2. 사용 변수

char\* str -- 함수의 인자로, 검사할 문자열이 저장되어있다.

int key -- 함수의 인자로, 어떤 유형의 문자열을 검사할지 정한다.

int i, t - 반복문의 인덱스, temp 등에 사용되는 임시 변수이다.

int flag - 한번이라도 플래그가 세워지면 해당 입력은 유효하지 않은 명령어이다.

int len - 문자열의 길이를 저장한다.

3. 12. 모듈 이름 : **add\_history(char \*str)**

3. 12. 1. 기능

정상적인 명령어일경우 새로운 노드를 생성하여 history list에 추가한다.

3. 12. 2. 사용 변수

char\* str -- 함수의 인자로, 사용자가 입력한 전체 명령어가 저장되어있다.

his\* temp - history 리스트를 탐색하기 위한 temp 노드이다.

his\* node - 리스트레 새로 추가하는, 정보가 저장된 노드이다.

3. 13. 모듈 이름 : **print\_history()**

3. 13. 1. 기능

history기능이 구현되어있다. 현재까지 저장된 history를 모두 출력한다.

3. 13. 2. 사용 변수

int cnt - 순서를 출력할 때 사용된다. 오래된 명령어부터 순서가 매겨진다.

his\* temp - history 리스트를 탐색하기 위한 temp 노드이다.

3. 14. 모듈 이름 : **free\_history()**

3. 14. 1. 기능

동적할당한 history list의 노드들을 모두 free시켜준다.

3. 14. 2. 사용 변수

his\* temp, prev - 리스트의 각 노드를 가리키는 temp 노드이다.

3. 15. 모듈 이름 : **free\_opcode()**

3. 15. 1. 기능

동적할당한 opcode list의 노드들을 모두 free시켜준다.

3. 15. 2. 사용 변수

hash\* temp, prev - 리스트의 각 노드를 가리키는 temp 노드이다.

3. 16. 모듈 이름 : **instruction\_check(char\* inst)**

3. 16. 1. 기능

입력받은 것이 어떤 명령어인지, 입력된 인자들은 정상적인 범위를 가지고 있는지 등을 체크한다. 또한 그러한 인자들을 배열에 따로 옮겨적는 수행도 한다. 유효하지 않은 명령어일경우 -1을 반환한다.

3. 16. 2. 사용 변수

char\* inst -- 함수의 인자로, 사용자가 입력한 명령어 전체가 저장된 문자열이다.

int flag - 명령어의 인자를 처리하기 위해 사용되는 플래그이다.

int i, idx - 반복문의 인덱스에 사용되는 임시 변수이다.

int len - 문자열의 길이를 저장하는 변수이다.

1. 전역 변수 정의
   1. int m[65536][16]

1MByte 크기의 가상 메모리 배열이다. 배열 하나당 1Byte크기의 0~FF의 범위를 가지며 해당 코드가 10진수로 변환되어 저장되어있다.

* 1. char start[10], end[10], address[10], value[10]

명령어로부터 입력받는 start, end, address, value가 캐릭터 문자열로 저장되어있다.

* 1. char mne[10]

opcode 명령어에서 mnemonic의 정보가 저장되어 있다.

* 1. int cur\_add

dump 명령어에서 출력할 첫번째 address가 저장되어있다. dump가 실행될때마다 갱신된다.

* 1. int end\_flag

전체적인 프로그램을 제어하는 플래그이다. quit 명령어를 입력받거나 파일열기에 실패할 경우 1로 플래그가 세워지며 프로그램은 종료된다.

* 1. int cur\_i

dump 명령어에서 출력할 첫번째 address의 index가 저장된 변수이다. 마찬가지로 dump가 실행될때마다 갱신된다.

* 1. hash\* h\_table[20]

opcode와 그 명령어, 형식에 대한 정보가 리스트 형태로 저장된 hash table의 head 배열이다.

* 1. his\* history

유효한 명령어가 리스트 형태로 저장되어있는 history list의 head이다.

1. 코드
   1. 20181618.h

#ifndef \_20181618

#define \_20181618

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys /stat.h>

#include <unistd.h>

#include <dirent.h>

// --------- opcode 표 정리하기 위한 구조체 ---------

typedef struct hash{

char mnemonic[10];

char op[10];

int type;

struct hash \*next;

}hash;

// --------- 명령어 history를 구성하기 위한 구조체 ---------

typedef struct his{

char inst[100];

struct his \*next;

}his;

// =========opcode.txt 파일 열고 읽는 함수=========

// 파일의 정보를 읽고 리스트에 저장한다.

// 반환값은 없다.

// =========================================

void readfile();

// =========hash 함수=========

// key는 명령의 첫번째 알파벳

// (key-'A')%21의 값으로 인덱싱하여

// value를 반환한다.

// ==========================

char \* hash\_function(char \* str);

// =========현재 디렉토리의 파일 출력=========

// 현재 디렉토리의 경로를 추출하여

// 그 안의 모든 파일을 출력한다.

// ==================================

void show\_directory();

// =========16진수->10진수 변환 함수=========

// 문자열 또는 문자를 받았을 때 10진수로 반환한다.

// ====================================

int hexTodec(char \* str, char c, int key);

// =========메모리 reset=========

// 전제 메모리 배열을 0으로 초기화한다.

// ===========================

void mem\_reset();

// =========10진수->16진수=========

// 인자로 들어온 값을 16진수 문자로 반환한다.

// =============================

char decTohec(int d);

// =========Dump 실행하는 함수=========

// 사용자가 지정한 범위의 메모리를 출력한다.

// 명령어의 종류에 따른 key값을 인자로 받고

// 결과를 출력한다.

// ===============================

void Dump(int key);

// =========edit 실행하는 함수.=========

// 입력받은 특정 주소의 값을 수정한다.

// ================================

void edit();

// =========fill 실행하는 함수=========

// 지정된 범위의 메모리 값을 수정한다

// ===============================

void fill();

// =========입력받은 값의 범위를 검사하는 함수==========

// start, end, addresss와 value의 범위 검사

// ========================================

int boundary\_check(char \* str, int key);

// =========정상적인 명령어 history에 추가하는 함수=========

void add\_history(char \*str);

// =========history 출력하는 함수==========

void print\_history();

// ===========history list free===========

void free\_history();

// ==========opcode list free===========

void free\_opcode();

// =========명령어가 정상적인지 결정하는 함수==========

// 어떤 명령어인지, 입력받은 값은 무엇인지, 그 값은 정상적인지

// 검사한 결과를 반환한다.

// ========================================

int instruction\_check(char \* inst);

int m[65536][16]; // 1MByte 메모리

char start[10], end[10], address[10], value[10]; // 입력받는 주소 및 value

char mne[10]; // opcode 명령어

int cur\_add; // 현재 출력할 address

int end\_flag; // 프로그램의 종료를 기록하는 플래그

int cur\_i; // 현재 출력할 address의 index

hash\* h\_table[20]; // hash table

his\* history; // history를 저장하는 링크드리스트의 head

#endif

* 1. 20181618.c

#include <stdio.h >

#include <string.h >

#include <stdlib.h >

#include <sys /types.h >

#include <sys /stat.h >

#include <unistd.h >

#include <dirent.h >

#include "20181618.h“

int main(){

int i;

int k\_inst =0;

hash\* temp;

char inst[50];

char \* t\_str;

readfile(); // opcode.txt를 읽고 저장

mem\_reset(); // memory 초기화

history = (his \*)malloc(sizeof(his));

history->next = NULL; // 명령어 history list의 head 초기화

cur\_add =0; end\_flag =0; cur\_i =0;

while(end\_flag == 0){ // end\_flag가 1이 될때까지 반복

printf("sicsim> ");

fgets(inst, 50, stdin);

inst[strlen(inst)-1] ='\0';

k\_inst = instruction\_check(inst); // 올바른 명령어인지, 어떤 명령어인지 반환받는다

if(k\_inst !=-1)

add\_history(inst); // 올바른 명령어일경우 history에 추가

switch(k\_inst){

case 0: // help

printf("h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start, end]\ne[dit] address, value\n");

printf("f[ill] start, end, value\nreset\nopcode mnemonic\nopcodelist\n");

break;

case 1: // dir

show\_directory();

break;

case 2: // quit

end\_flag =1;

break;

case 3: // history

print\_history();

break;

case 41: // dump

Dump(1);

break;

case 42: // dump start

Dump(2);

break;

case 43: // dump start, end

Dump(3);

break;

case 5: // dump address, value

edit();

break;

case 6: //fill start, end, value

fill();

break;

case 7: // reset

mem\_reset();

break;

case 8: // opcode

t\_str = hash\_function(mne);

if(strcmp(t\_str, "\0") ==0) printf(" not exist!\n"); // 리스트에 없을때

else printf("opcode is %s\n", t\_str);

break;

case 9: // opcode list

for(i =0; i <20;i ++){

printf(" %d ",i);

temp = h\_table[i]->next;

while(temp != NULL){

printf("-> [%s,%s] ", temp ->mnemonic, temp ->op);

temp = temp ->next;

}

printf("\n");

}

break;

default:

break;

}

}

return 0;

}

// =========opcode.txt 파일 열고 읽는 함수=========

// 파일의 정보를 읽고 리스트에 저장한다.

// 반환값은 없다.

// =========================================

void readfile(){

FILE\* f = fopen("opcode.txt","r");

char buf[25];

int i, idx, flag =0;

int cnt =0;

int key =0;

if(f ==NULL){ // 파일 오픈 실패

printf(" file open error!\n");

end\_flag =1;

return ;

}

for(i =0;i <20;i ++){

h\_table[i] = (hash \*)malloc(sizeof(hash));

}

hash \*temp;

while(cnt <58){

fgets(buf, 25, f); // 한 줄 읽기

hash \*node = (hash \*)malloc(sizeof(hash));

node->next = NULL;

node->op[0] = buf[0]; node ->op[1] = buf[1]; // opcode 저장

i=1; idx =0; flag =0;

while(1){ // 명령어 저장

i++;

if(buf[i] >='A'&& buf[i] <='Z'){

node->mnemonic[idx ++] = buf[i];

flag =1;

}

if(flag ==1 && buf[i] =='\t') break;

}

key = (node ->mnemonic[0]-'A')%21;

temp = h\_table[key];

while(temp ->next != NULL){

temp = temp ->next;

}

temp->next = node;

cnt++;

}

fclose(f);

}

// =========hash 함수=========

// key는 명령의 첫번째 알파벳

// (key-'A')%21의 값으로 인덱싱하여

// value를 반환한다.

// ==========================

char \* hash\_function(char \* str){

int key = (str[0]-'A')%21; // key는 입력받은 명령어의 첫번째 알파벳

// 배열의 index는 알파벳 순서로 하되,

// W는 20을 넘어가므로 %21를 통해 인덱싱한다.

hash\* temp = h\_table[key];

while(temp != NULL){

if(strcmp(str, temp ->mnemonic) ==0){

return temp ->op;

}

temp = temp ->next;

}

return "\0"; // 검색결과 존재하지 않을경우 NULL

}

// =========현재 디렉토리의 파일 출력=========

// 현재 디렉토리의 경로를 추출하여

// 그 안의 모든 파일을 출력한다.

// ==================================

void show\_directory(){

DIR\* dir\_p = NULL;

struct dirent \* entry = NULL;

struct stat buf;

char \*path;

int cnt =0;

path = getcwd(NULL, 500); // 현재 디렉토리의 절대경로 추출

dir\_p = opendir(path); // 경로를 통해 디렉토리를 open한다

while((entry = readdir(dir\_p)) != NULL){ // 폴더 안의 모든 파일들을 검색한다

lstat(entry->d\_name, &buf);

if(S\_ISDIR(buf.st\_mode)) printf("%24s/", entry ->d\_name); // 디렉토리 파일

else if(S\_IEXEC & buf.st\_mode) printf("%24s\*", entry ->d\_name); // 실행 파일

else printf("%25s", entry ->d\_name); // 그 외 파일

cnt++;

if(cnt%4 ==0) printf("\n");

}

if(cnt%4 !=0) printf("\n");

closedir(dir\_p);

}

// =========16진수->10진수 변환 함수=========

// 문자열 또는 문자를 받았을 때 10진수로 반환한다.

// ====================================

int hexTodec(char \* str, char c, int key){

int sum, h;

if(key ==0){ // 문자열이 input일때

int len = strlen(str);

int i, a =1;

for(i =0;i <len -1;i ++){ // 전체 자릿수 맞추기

a \*=16;

}

sum =0;

for(i =0;i <len;i ++){

if(str[i] >='0'&& str[i] <='9')

h = str[i] -'0';

else if(str[i] >='a'&& str[i] <='z')

h = str[i] -'a'+10;

else if(str[i] >='A'&& str[i] <='Z')

h = str[i] -'A'+10;

sum += h \*a;

a /=16;

}

return sum;

}

else{ // char이 input일때

if(c >='0'&& c <='9')

h = c -'0';

else if(c >='a'&& c <='z')

h = c -'a'+10;

else if(c >='A'&& c <='Z')

h = c -'A'+10;

return h;

}

}

// =========메모리 reset=========

// 전제 메모리 배열을 0으로 초기화한다.

// ===========================

void mem\_reset(){

int i, j;

for(i =0;i <65536;i ++){

for(j =0;j <16;j ++){

m[i][j] =0;

}

}

}

// =========10진수->16진수=========

// 인자로 들어온 값을 16진수 문자로 반환한다.

// =============================

char decTohec(int d){

if(d >=0 && d <=9){

return d +'0';

}

else{

return d -10 +'A';

}

}

// =========Dump 실행하는 함수=========

// 사용자가 지정한 범위의 메모리를 출력한다.

// 명령어의 종류에 따른 key값을 인자로 받고

// 결과를 출력한다.

// ===============================

void Dump(int key){

int i, j, a, temp, t; // temp 및 index 변수

int st\_i, st\_j, cnt, st\_n; // st\_i, st\_j - 출력 시작할 메모리의 인덱스

int show\_addr[5] = { 0, }; // 주소 출력할 배열

char t\_str[4]; // temp 배열

if(key ==1){ // du[mp] 명령어 처리

st\_i =0;

for(i =cur\_i;i <10 +cur\_i;i ++){

temp = cur\_add;

a =16 \*16 \*16 \*16;

for(j =0;j <4;j ++){ // 출력할 주소 앞의 4자리 계산

show\_addr[j] = temp /a;

temp -= (temp /a)\*a;

a /=16;

}

for(j =0;j <5;j ++){ // 주소 출력

printf("%c", decTohec(show\_addr[j]));

}

printf(" ");

for(j =0;j <16;j ++){ // 메모리 출력. 저장된 정수값을 16진수로 변환한다.

t = m[i][j]/16;

printf("%c%c ", decTohec(t), decTohec(m[i][j]-t \*16));

}

printf(" ; ");

for(j =0;j <16;j ++){ // ASCII코드 출력

if(m[i][j]>=32 && m[i][j]<=126) // 범위는 20 ~ 7E

printf("%c", m[i][j]);

else

printf(".");

}

printf("\n");

cur\_add +=16; // 다음에 출력할 첫번째 주소값

if(i >=65535){ // FFFFF를 넘어갈 경우 초기화

cur\_i =-10;

cur\_add =0;

}

}

cur\_i +=10; // m[i][] index 갱신

}

else if(key ==2){ // du[mp] start 명령어 처리

cnt =0;

temp = hexTodec(start, '0', 0);

st\_n = temp;

while(cnt !=160){ // 입력받은 start부터 160개 출력

a =16 \*16 \*16 \*16;

for(j =0;j <4;j ++){

show\_addr[j] = temp /a;

t\_str[j] = decTohec(temp /a);

temp -= (temp /a)\*a;

a /=16;

}

t\_str[4] ='\0';

st\_i = hexTodec(t\_str, '0', 0); // [st\_i][]부터 출력 시작

st\_j = temp; //[][st\_j]부터 출력 시작

for(j =0;j <5;j ++){ // 주소 출력

printf("%c", decTohec(show\_addr[j]));

}

printf(" ");

if(cnt ==0){ // 첫번째 줄

i = st\_i;

for(j =0;j <st\_j;j ++){ // 출력되지 않는 부분은 공백처리

printf(" ");

}

for(j =st\_j;j <16;j ++){

t = m[i][j]/16;

printf("%c%c ", decTohec(t), decTohec(m[i][j]-t \*16));

cnt++;

}

printf(" ; ");

for(j =0;j <st\_j;j ++){ // 출력되지 않는 ASCII코드는 '.'처리

printf(".");

}

for(j =st\_j;j <16;j ++){

if(m[i][j]>=32 && m[i][j]<=126)

printf("%c", m[i][j]);

else

printf(".");

}

}

else if(160 -cnt <16){ // 마지막 줄

i = st\_i;

st\_j =160 - cnt;

for(j =0;j <st\_j;j ++){

t = m[i][j]/16;

printf("%c%c ", decTohec(t), decTohec(m[i][j]-t \*16));

cnt++;

}

for(j =st\_j;j <16;j ++){ // 출력되지 않는 부분은 공백처리

printf(" ");

}

printf(" ; ");

for(j =0;j <st\_j;j ++){

if(m[i][j]>=32 && m[i][j]<=126)

printf("%c", m[i][j]);

else

printf(".");

}

for(j =st\_j;j <16;j ++){ // 출력되지 않는 ASCII코드는 '.'처리

printf(".");

}

}

else{ // 전체 출력되는 줄

i = st\_i;

for(j =0;j <16;j ++){

t = m[i][j]/16;

printf("%c%c ", decTohec(t), decTohec(m[i][j]-t \*16));

cnt++;

}

printf(" ; ");

for(j =0;j <16;j ++){

if(m[i][j]>=32 && m[i][j]<=126)

printf("%c", m[i][j]);

else

printf(".");

}

}

printf("\n");

if(i ==65535) break; // 주소값 FFFFF까지 출력될 경우 멈춘다.

temp = st\_n + cnt; // 출력한 만큼 주소값 갱신

}

}

else{ // du[mp] start, end 명령어 처리

cnt =0;

temp = hexTodec(start, '0', 0);

st\_n = temp;

int max = hexTodec(end, '0', 0) - temp +1; // 출력될 byte 숫자 계산

while(cnt != max){

a =16 \*16 \*16 \*16;

for(j =0;j <4;j ++){

show\_addr[j] = temp /a;

t\_str[j] = decTohec(temp /a);

temp -= (temp /a)\*a;

a /=16;

}

t\_str[4] ='\0';

st\_i = hexTodec(t\_str, '0', 0); // [st\_i][]부터 출력 시작

st\_j = temp; //[][st\_j]부터 출력 시작

for(j =0;j <5;j ++){ // 주소 출력

printf("%c", decTohec(show\_addr[j]));

}

printf(" ");

if(max <=16){ // 출력할 byte 수가 한 줄 이하일 경우

i = st\_i;

for(j =0;j <st\_j;j ++){

printf(" ");

}

j = st\_j;

while(cnt != max){ // 메모리 출력

t = m[i][j]/16;

printf("%c%c ",decTohec(t), decTohec(m[i][j]-t \*16));

cnt++; j ++;

}

for(;j <16;j ++){

printf(" ");

}

printf(" ; ");

for(j =0;j <st\_j;j ++){

printf(".");

}

cnt=0;

while(cnt != max){ // ASCII코드 출력

if(m[i][j]>=32 && m[i][j]<=126)

printf("%c", m[i][j]);

else

printf(".");

j++; cnt ++;

}

for(;j <16;j ++){

printf(".");

}

}

else if(cnt ==0){ // 첫번째 줄

i = st\_i;

for(j =0;j <st\_j;j ++){

printf(" ");

}

for(j =st\_j;j <16;j ++){ // 메모리 출력

t = m[i][j]/16;

printf("%c%c ", decTohec(t), decTohec(m[i][j]-t \*16));

cnt++;

}

printf(" ; ");

for(j =0;j <st\_j;j ++){

printf(".");

}

for(j =st\_j;j <16;j ++){ // ASCII코드 출력

if(m[i][j]>=32 && m[i][j]<=126)

printf("%c", m[i][j]);

else

printf(".");

}

}

else if(max -cnt <16){ // 마지막 줄

i = st\_i;

st\_j = max - cnt;

for(j =0;j <st\_j;j ++){ // 메모리 출력

t = m[i][j]/16;

printf("%c%c ", decTohec(t), decTohec(m[i][j]-t \*16));

cnt++;

}

for(j =st\_j;j <16;j ++){

printf(" ");

}

printf(" ; ");

for(j =0;j <st\_j;j ++){ // ASCII코드 출력

if(m[i][j]>=32 && m[i][j]<=126)

printf("%c", m[i][j]);

else

printf(".");

}

for(j =st\_j;j <16;j ++){

printf(".");

}

}

else{ // 전체 출력되는 줄

i = st\_i;

for(j =0;j <16;j ++){ // 메모리 출력

t = m[i][j]/16;

printf("%c%c ", decTohec(t), decTohec(m[i][j]-t \*16));

cnt++;

}

printf(" ; ");

for(j =0;j <16;j ++){ // ASCII코드 출력

if(m[i][j]>=32 && m[i][j]<=126)

printf("%c", m[i][j]);

else

printf(".");

}

}

printf("\n");

temp = st\_n + cnt; // 주소값 갱신

}

}

}

// =========edit 실행하는 함수.=========

// 입력받은 특정 주소의 값을 수정한다.

// ================================

void edit(){

int i, j, a;

int temp = hexTodec(address, '0', 0); // 입력받은 주소값 10진수로 저장

char t\_str[4];

a =16 \*16 \*16 \*16;

for(j =0;j <4;j ++){

t\_str[j] = decTohec(temp /a);

temp -= (temp /a)\*a;

a /=16;

}

t\_str[4] ='\0';

i = hexTodec(t\_str,0,0); // 주소값 앞의 4자리를 10진수로 바꾼 값이 i

j = temp; // 주소값 맨 끝자리를 10진수로 바꾼 값이 j

m[i][j] = hexTodec(value, 0, 0); // 입력받은 value 저장

}

// =========fill 실행하는 함수=========

// 지정된 범위의 메모리 값을 수정한다

// ===============================

void fill(){

int a, i, j, st\_i, st\_j, st\_n;

int cnt, max, temp;

char t\_str[4];

cnt =0;

temp = hexTodec(start, 0, 0);

st\_n = temp;

max = hexTodec(end, 0, 0) - temp +1; // 출력될 byte 숫자 계산

while(cnt != max){

a =16 \*16 \*16 \*16;

for(j =0;j <4;j ++){

t\_str[j] = decTohec(temp /a);

temp -= (temp /a)\*a;

a /=16;

}

t\_str[4] ='\0';

st\_i = hexTodec(t\_str, 0, 0); // 주소값 앞의 4자리를 10진수로 바꾼 값이 i

st\_j = temp; // 주소값 맨 끝자리를 10진수로 바꾼 값이 j

if(max <=16){ // 갱신할 메모리가 한 줄 이하일때

i = st\_i;

j = st\_j;

while(cnt != max){

m[i][j] = hexTodec(value,0,0);

cnt++; j ++;

}

}

else if(cnt ==0){ // 첫번째 줄

i = st\_i;

for(j =st\_j;j <16;j ++){

m[i][j] = hexTodec(value, 0, 0);

cnt++;

}

} // 마지막 줄

else if(max -cnt <16){

i = st\_i;

st\_j = max - cnt;

for(j =0;j <st\_j;j ++){

m[i][j] = hexTodec(value, 0, 0);

cnt++;

}

}

else{ // 전체 출력되는 줄

i = st\_i;

for(j =0;j <16;j ++){

m[i][j] = hexTodec(value, 0, 0);

cnt++;

}

}

temp = st\_n + cnt; // 주소값 갱신

}

}

// =========입력받은 값의 범위를 검사하는 함수==========

// start, end, addresss와 value의 범위 검사

// ========================================

int boundary\_check(char \* str, int key){

int flag =0;

int t,i;

int len = strlen(str);

for(i =0;i <len;i ++){ // 입력받은 인자는 모두 16진수이다.

t = str[i];

if(t >='0'&& t <='9'){}

else if(t >='a'&& t <='f'){}

else if(t >='A'&& t <='F'){}

else flag =1;

if(key ==0){ // start, end, address의 경우

if(len >5) flag =1;

}

else if(key ==1){ // value의 경우

if(len >2) flag =1;

}

}

if(flag ==1) return 1; // 한 번 이라도 플래그가 세워지면 비정상적인 input

else return 0; // 정상적인 경우

}

// =========정상적인 명령어 history에 추가하는 함수=========

void add\_history(char \*str){

his\* temp = history;

his\* node = (his \*)malloc(sizeof(his));

node->next = NULL;

strcpy(node->inst, str); // 노드에 명령어 저장

while(temp ->next != NULL){

temp = temp ->next;

}

temp->next = node;

}

// =========history 출력하는 함수==========

void print\_history(){

int cnt =1;

his\* temp = history ->next;

while(temp !=NULL){

printf("%10d %s\n", cnt ++, temp ->inst);

temp = temp ->next;

}

}

// =========history list free==========

void free\_history(){

his\* temp = history;

his\* prev = NULL;

while(temp->next != NULL){

prev = temp;

temp = temp -> next;

free(prev);

}

free(temp);

}

// =========opcode list free==========

void free\_opcode(){

hash\* temp;

hash\* prev = NULL;

for(int i=0;i<20;i++){

temp =h\_table[i];

while(temp->next != NULL){

prev = temp;

temp = temp->next;

free(prev);

}

free(temp);

}

}

// =========명령어가 정상적인지 결정하는 함수==========

// 어떤 명령어인지, 입력받은 값은 무엇인지, 그 값은 정상적인지

// 검사한 결과를 반환한다.

// ========================================

int instruction\_check(char \* inst){

int flag =0;

int i, idx, len =0;

for(i =0;i <10;i ++){ // 배열 초기화

start[i] ='\0';

end[i] ='\0';

address[i] ='\0';

value[i] ='\0';

mne[i] ='\0';

}

if(strcmp(inst, "help") ==0 || strcmp(inst, "h") ==0){

return 0;

}

else if(strcmp(inst, "dir") ==0 || strcmp(inst, "d") ==0){

return 1;

}

else if(strcmp(inst, "quit") ==0 || strcmp(inst, "q") ==0){

return 2;

}

else if(strcmp(inst, "history") ==0 || strcmp(inst, "hi") ==0){

return 3;

}

else if(strncmp(inst, "dump", 4) ==0 || strncmp(inst, "du", 2) ==0){

len = strlen(inst);

idx=0; i =0; flag =0;

if(strcmp(inst, "dump") ==0 || strcmp(inst, "du") ==0)

return 41; // du[mp]만 입력될 경우

for(i =0;i <=len;i ++){

if(flag ==0 && inst[i] ==' '){ // ' '이 있으면 start는 있는 명령어

flag =1;

i++;

}

else if(inst[i] ==','){ // ','가 있으면 start, end가 입력된 명령어

flag =2;

idx =0; i +=2;

}

if(flag ==1){

start[idx++] = inst[i];

}

else if(flag ==2){

end[idx++] = inst[i];

}

if(flag ==1 && i == len){ // start 받았는데 문장이 끝날 경우

if(boundary\_check(start, 0) ==1){

printf(" input error!\n");

return -1;

}

return 42; // start의 값 체크하고 검사 결과 return

}

else if(flag ==2 && i == len){ // start, end 받고 문장 끝날 경우

if(boundary\_check(start, 0) ==1 || boundary\_check(end, 0) ==1){

printf(" input error!\n");

return -1;

}

if(hexTodec(start,0,0)>hexTodec(end,0,0)){

printf(" start>end!\n");

return -1;

}

return 43; // start, end와 범위 체크하고 검사 결과 return

}

}

}

else if(strncmp(inst, "edit ", 4) ==0 || strncmp(inst, "e ", 1) ==0){

len = strlen(inst);

i =0; idx =0; flag =0;

for(i =0;i <len;i ++){

if(flag ==0 && inst[i] ==' '){ // 공백 이후 address 입력받기

flag =1;

i++;

}

else if(inst[i] ==','){ // ',' 이후 value 입력받기

flag =2;

idx =0; i +=2;

}

if(flag ==1){

address[idx++] = inst[i];

}

else if(flag ==2){

value[idx++] = inst[i];

}

}

if(boundary\_check(address,0) ==1 || boundary\_check(value, 1) ==1){

printf(" input error!\n");

return -1;

} // address, value 값 체크 후 결과 return

return 5;

}

else if(strncmp(inst, "fill", 4) ==0 || strncmp(inst, "f", 1) ==0){

len = strlen(inst);

flag=0; i =0; idx =0;

for(i =0;i <len;i ++){

if(flag ==0 && inst[i] ==' '){ // 첫번째 공백 이후 start입력받음

flag =1;

i++;

}

else if(flag ==1 && inst[i] ==','){ // 첫번째 ',' 이후 end 입력받음

flag =2;

idx =0; i +=2;

}

else if(flag ==2 && inst[i] ==','){ // 두번째 ',' 이후 value 입력받음

flag =3;

idx =0; i +=2;

}

if(flag ==1){

start[idx++] = inst[i];

}

else if(flag ==2){

end[idx++] = inst[i];

}

else if(flag ==3){

value[idx++] = inst[i];

}

}

if(boundary\_check(start,0) ==1 || boundary\_check(end, 0) ==1){

printf(" input error!\n");

return -1;

}

if(boundary\_check(value, 1) ==1){

printf(" input error!\n");

return -1;

}

if(hexTodec(start,0,0)>hexTodec(end,0,0)){

printf(" start>end!\n");

return -1;

}

return 6; // start, end, value 및 범위 검사 후 return

}

else if(strcmp(inst, "reset") ==0){

return 7;

}

else if(strncmp(inst, "opcode ", 7) ==0){

len = strlen(inst);

flag =0; idx =0;

for(i =0;i <len;i ++){

if(inst[i] ==' '){ // 공백 이후 명령어 받음

flag =1;

i++;

}

if(flag ==1) mne[idx ++] = inst[i];

}

return 8;

}

else if(strcmp(inst, "opcodelist") ==0){

return 9;

}

return -1; // 위의 경우 모두 만족하지 못하면 유효하지 않은 명령어

}